Babel 1.0 Release Criteria: A Working Document

December 2003

Overview

In keeping with the Open Source tradition, we want our Babel 1.0 release to indicate a certain level of capability, maturity, and stability. From our first release (version 0.5.0) in July of 2001 to our current (fourteenth) release (version 0.8.8) we have continued to add capabilities in response to customer feedback, our observations in the field, and a consistent vision for interoperability. The key to Babel's maturity is without a doubt the ever-increasing demands of our growing user base… both in terms of sheer size and sophistication with the underlying technology.

Stability is a special challenge for any research project. With our 1.0 release, we will maintain a stable Babel 1.0 code branch for at least a full year. Only backward compatible bugfixes will be applied to the branch. All continuing R&D will continue along the trunk. This means that Babel 1.0.x releases will likely be made *after* a Babel 1.1.0. Currently, Babel has a quarterly release cycle with no guarantee for backward compatibility from one release to the next.

Now is the time where we can see a good point for a Babel 1.0 release. But, seeing that point is different from being there. This list enumerates and explains the outstanding technical issues to be resolved to minimize volatility and help ensure stability for the 1.0 line.

The first draft of this document was circulated internally in June 2003. A revised draft was then presented at the July 2003 CCA meeting. This document represents the third revision intended for general comment by the babel-users group and to be posted on the Babel homepage for the public to track progress in subsequent Babel releases.

Table of Contents

1. *Parser*

a. Regularize Type Resolution Behavior

The Babel parser currently is too aggressive in resolving user-defined types. This causes problems with SIDL files needed to be listed in specific order on the command line. It also causes the following two SIDL files to be resolve the return type of foo.A.bar() differently. Clearly a lazy resolution technique is needed where type resolution is a distinct phase from initial parsing.

```
 package foo {
 class A {
  foo.B bar ();
 }
 class B {}
}
```

```
 package foo {
 class B {}
 class A {
  foo.B bar ();
 }
}
```

2. *SIDL*

Every SIDL grammar change listed is an addition to the current SIDL spec. The following changes should not invalidate existing code.

a. Add global scope indicator (.) (like leading :: in C++)

```
1. package foo {
2.    class A { }
3.     package foo {
4.       class A {
5.         foo.A bar(); // returns which foo.A?
6.       }
7.     }
8.   }
```

There are limited cases where presented with multiple options, SIDL gives no easy way to specify exactly which choice the user wants. In the SIDL example above, line 5 shows a method bar() that returns a foo.A. Currently, this resolves to the class in the same package, and its not possible to refer to the class A in the outermost package. Adding a leading dot (.) to the syntax resolves this issue. The user could specify ".foo.A" to resolve the ambiguity.

b. Resolve issues with multiple inheritance induced overloading

```
  interface I1 {
    void set( in int i );
  }
  interface I2 {
    void set( in float f );
  }

  class C implements-all I1, I2 { }
```

The SIDL fragment above is currently invalid. The multiple inheritance of interfaces would be fine if both set() methods had the same signature. However, since they are different and since they are inherited from different interfaces they are not overloaded. SIDL's grammar and Babel's parser need to be augmented to cover this case.

Tentatively the new syntax would look like the following SIDL fragment.

```
  class C implements I1, I2 {
    void set[Int](in int i) = I1.set;
    void set[Flt](in float f) = I2.set;
  }
```

Here, the equals operator allows us to redefine the name of a method that is inherited, but does not allow the signature to be changed.

3. IOR

Because the IOR is the key to Babel's interoperability, we expect each change listed here to be non-backward compatible with older IORs. Customers will have to run the newer Babel code generator over their SIDL files to generate new IORs.

a. Access super methods in Impls

Often times in the implementation of a method, it is common to want to call the parent class's implementation. Babel currently stores the Entry Point Vector (EPV) (similar to a C++ vtable) for the parent class as part of proper construction/destruction, but the EPV is hidden in a static variable in the IOR.c file, and thereby not exposed for the implementation to use.

We need to provide some equivalent to "super" methods in Java. Super methods are allow the implementation of a method in a derived class to invoke otherwise overridden methods in the parent (or super) class.

As of December 2003, this activity is in the initial design stage.

b. RMI hooks

Babel will define a SIDL interface standard for Remote Method Invocation (RMI) and will generate hooks in the IOR to call those interfaces. This will allow interested researchers with communication libraries to implement these interfaces and plug their code into Babel.

As of December 2003, there are some hand-generated prototypes using Ken Chiu's Proteus multi-protocol library from Indiana University.

c. Pre/post method hooks

There are many cases where one may want to hook arbitrary code as a precondition or postcondition of a Babel method invocation. Examples include logic checking, timer insertion, flow traces, and QoS. Babel will define and implement a general standard to satisfy the community's interest in this feature.

As of December 2003, we have received requirements from the TAU team for instrumenting Babel code with timers. A draft proposal is in the design stages and will be circulated to the babel-users list for comment.

4. Runtime

a. Better shared library lookup in SIDL.Loader

When asked to load a symbol at runtime, the original SIDL.Loader would go through its path, recursively opening every .so file looking for the symbol in question. If the user set their SIDL_DLL_PATH to "/", the SIDL.Loader would find it… eventually. The problems with this all too permissive approach were (1) it was hard to debug, (2) it would try to open things that had a .so extension, but were not made to be dynamically loaded. (3) it was too hard to control.

The SIDL.Loader implementation will be rewritten to be must less permissive and easier to control and debug. The new implementation will rely on auxiliary XML files to specify exactly what symbols are to be found in what libraries.

As of December 2003, the SIDL.Loader is currently implemented and introduces non-backward compatible changes. It will be released in the upcoming Babel 0.9.0 release (expected in Q1 2004)

b. Move Base Classes from "SIDL" package to "sidl" package

Unfortunately some C/C++ header files on some architectures #define SIDL to be 4. This causes problems in C++ header files that put everything in the namespace SIDL. We have always recommended that SIDL package names be all in lowercase, but for historical reasons we haven't followed our own advice. Since the convention is that preprocessor macros are always all uppercase, we are resolved to rename the SIDL package to lowercase.

This change should avoid preprocessor problems, but will introduce a non-backward compatible change when released. As of December 2003, there has been no work on this item.

c. Change SIDL.BaseException from a class to an Interface

This is of primary importance to standards bodies, such as CCA who want to make their standard be all SIDL interfaces, but currently have to make the exceptions SIDL classes because SIDL.BaseException is a class. This is an artifact that we unintentionally carried from Java and we plan to be correct it.

As of December 2003, there has been some initial design work. We have uncovered issues with Java bindings in response to this change that have yet to be resolved.

~~5.~~ ~~*Arrays*~~

a. ~~Increase max dimension from 4 to 7~~

DONE. Released in Babel 0.8.8

6. *C++*

Below are two nontrivial shortcomings with our current C++ bindings. Interested in using Babel in SCIRun, Steven Parker from University of Utah will be visiting LLNL the last week of January 2004 to help us fix these problems. At this point, a complete rewrite of the C++ bindings cannot be ruled out. If a complete rewrite is needed, a suitable migration path will be a BIG DEAL for our customers. We'll have a better feel for what to do, after we see how much the C++ bindings change to address these issues.

a. Resolve issues with overloading based on object type

Babel's C++ bindings predate Babel's overloading mechanism. The C++ stubs simulate the SIDL inheritance hierarchy, but do not implement it directly. The benefit is that method dispatch is slightly faster by circumventing C++ vtables and going directly to Babel's EPVs. The downside is that once overloading was introduced, the C++ compiler doesn't have the right information for native C++ overloading to work in all cases.

b. Resolve issues with exception handling of Babel smart-pointers

The issue is that the C++ code cannot catch parent classes of the SIDL declared exception, it must catch the exceptions listed in the SIDL specification exactly. This is another unfortunate side-effect of Babel's current C++ stubs.

7. *Java*

As of December 2003, we now have Java bindings that support all the basic SIDL types (bool, char, int, float, long, double, fcomplex, dcomplex, and strings), both client side and server side. However, arrays and objects remain outstanding issues. Arrays and Objects are historically the most challenging features of any language binding. The reasons for the delays in Java is that Fortran became a very big deal for our customers and Java was put on the back-burner.

a. Implement Array support (client and server)

Native Java arrays are not suitable for our purposes (appear to require copying data in all cases). Python had a similar problem, but there was a single we-established array library (Numeric) that was a natural fit. We have yet to find a similarly obvious choice for scientific arrays in Java.

b. Implement support for Objects as arguments (client and server)

There seems to be no outstanding technical issues to resolve, it is simply a matter of finding an available pool of effort to throw at the problem.

8. *F90*

~~a. Provide native access to Babel arrays (a.k.a. Phase III)~~

DONE. Released in Babel 0.8.8. Introduced new dependency on Chasm.

b. Find solution for compilers that treat intrinsic functions as reserved words

Some compilers treat intrinsic functions as reserved words. Although Babel does try to warn when reserved words in a particular language show up in a SIDL specification, we think prohibiting all the Fortran 90 intrinsic functions is a bit onerous. We hope to find a workaround before the Babel 1.0 release, but this issue will not be a blocker for the 1.0 release.

9. *Documentation*

a. More detail on SIDL language

We will add an annotated EBNF specification of the SIDL grammar and walk through more techniques about how to transfer favorite programming idioms from different languages to SIDL.

b. More Examples

Our Users' Guide has been described as encyclopedic and very dense. Users have asked for more examples to reduce the slope of the learning curve in the first couple sections.

c. More Tutorials

Babel has developed a half-day tutorial and delivered a dry-run to the CASC summer interns in 2003. We will be polishing the viewgraphs, presentations, and demonstrations for the 1.0 release.

10. Platforms

~~a. AIX using native xl Compilers~~

DONE (mostly). Official support announced in Babel 0.8.6. Unresolved issues with server-side python support.

~~b. Revivify Solaris using gcc/SunF90~~

DONE. Will be included in next release after Babel 0.8.8.

c. Linux64

DONE (mostly). Works with Babel 0.8.9, but not instituted in nightly testing. Also required a lot of rebuilding of basic tools (e.g., bintools)?

d. Others ?

Deferred Items

The following items are frequently requested features. We know only that they are not planned for Babel 1.0. There's no telling if they will ever get into Babel for that matter, but we haven't removed them entirely from future considerations either.

1. Adding type hierarchy information to sidl.ClassInfo
2. Change from Numeric Python to numarray
3. Stop generating IOR.c files with client-side binding
4. Typed Opaques
5. Array Base Class(es)